VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Vývoj mobilní aplikace pro ČAHD

Development of Mobile Application for ČAHD

2014 Jakub Vašica

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jakub Vašica**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Vývoj mobilní aplikace pro ČAHD
Development of Mobile Application for ČAHD

Zásady pro vypracování:

V rámci bakalářské práce student prostuduje možnosti multiplatformního vývoje aplikací pro mobilní zařízení. Student vybere vhodný framework pro vývoj aplikací pro více mobilních operačních systému jakými jsou iOS, Android či Windows Phone. Pomocí vybraného frameworku student provede implementaci aplikace určené pro využití Českou asociací hasičských důstojníků (ČAHD).

Jednotlivé body zadání jsou:
1. Průzkum a porovnání frameworků pro vývoj multiplatformních aplikací.
2. Tvorba aplikace ve vybraném frameworku – aplikace bude načítat XML soubory a podle nich dynamicky generovat GUI prvky a vizualizovat obsah XML.
3. Dále bude aplikace podporovat interaktivní režim, ve kterém bude možno ukládat vybrané informace na vzdálené datové uložiště a vyčítat GPS pozici mobilního zařízení.
4. Provedení funkčních testů na zařízeních s operačními systémy iOS a Android.

Seznam doporučené odborné literatury:

[1] Titanium SDK: http://www.appcelerator.com/titanium/titanium-sdk/
[2] Android SDK: http://developer.android.com/sdk/index.html
[3] iOS Dev Center: https://developer.apple.com/devcenter/ios/index.action

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Martinovič, Ph.D.**

Datum zadání: 01.09.2013
Datum odevzdání: 07.05.2014

VYSOKÁ ŠKOLA BÁŇSKÁ - TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA *

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 29. dubna 2014 .....

Rád bych na tomto místě poděkoval svému vedoucímu panu Ing. Janu Martinovičovi Ph.D.
za rady a konzultace při vytváření této práce.

Abstrakt

Práce se zabývá průzkumem multiplatformních frameworků pro vývoj mobilních aplikací a vývojem mobilních aplikací v JavaScriptovém nástroji Appcelerator Titanium. Vyvíjeny jsou dvě podobné aplikace na Android a iOS z oblasti požární ochrany pro Českou asociaci hasičských důstojníků. Aplikace využívají digitalizovaných XML dokumentů, na jejichž základě se dynamicky generuje grafické uživatelské rozhraní. První aplikace se nazývá Vaše cesty k bezpečí a jejím účelem je poskytovat rady v krizových situacích. Druhá aplikace se jmenuje Pomůcka velitele jednotky požární ochrany a umožňuje uživateli zadávat naměřené údaje, zachytávat polohu GPS a odesílat soubory do vzdáleného datového uložiště Dropbox.

Klíčová slova: multiplatformní, mobilní aplikace, Titanium Appcelerator

Abstract

The thesis researches cross-platform frameworks for mobile application development and also describes the development of mobile applications in JavaScript-based tool Appcelerator Titanium. Two similar applications with the topic of fire protection are developed for Android and iOS on behalf of The Czech Fire-fighters Union. Applications dynamically generate graphical user interface, which is based on digitalized XML documents. The first application is called Your Path to the Safety and it is supposed to advise people how to deal with crisis situations. The second application is called The Aid for the Chief Fire Officer and it allows the user to enter measured values, find GPS coordinates and send files to the remote storage Dropbox.

Keywords: cross-platform, mobile application, Titanium Appcelerator

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
BLOB	–	Binary large object
CSS	–	Cascading Style Sheets
ČAHD	–	Česká asociace hasičských důstojníků
DIP	–	Density Independent Pixels
DOM	–	Document Object Model
GPS	–	Global Positioning System
GUI	–	Graphical User Interface
HTML	–	Hyper Text Markup Language
JS	–	JavaScript
JSS	–	JavaScript Style Sheets
IDE	–	Integrated Development Environment
iOS	–	iPhone Operating System
MVC	–	Model View Controller
OS	–	Operating System
SDK	–	Software Development Kit
UI	–	User Interface
URL	–	Uniform Resource Locator
W3C	–	World Wide Web Consortium
WP	–	Windows Phone
XML	–	Extensible Markup Language

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

V dnešní době zažívají mobilní zařízení neskutečný rozvoj. Prodeje stolních počítačů a notebooků klesají a stále více lidí si kupuje raději tablet nebo chytrý telefon. S narůstajícím počtem mobilních zařízení roste poptávka po aplikacích z různých oblastí, mimo jiné z oblasti požární ochrany. Na stolních počítačích má majoritní podíl operační systém (OS) Windows, takže vývoj aplikací nečiní přílišné problémy, kdežto na mobilních zařízeních je podíl operačních systémů značně roztroušen, čímž se náročnost vývoje komplikuje.

Cílem práce je vyvinout dvě aplikace pro Českou asociaci hasičských důstojníků, a sice informační aplikaci pro veřejnost a pomocnou aplikaci pro hasičský sbor. Aplikace mají fungovat na více mobilních operačních systémech, nabízí se proto možnost multiplatformního vývoje – vytvořit univerzální aplikaci pro více platforem.

V následující kapitole se podíváme na multiplatformní vývoj mobilních aplikací, některé vybrané nástroje a jejich možnosti. Třetí kapitola je věnována vývoji aplikace *Vaše cesty k bezpečí*. Obsahuje informace o zdrojových datech, multiplatformních technikách tvorby grafického uživatelského rozhraní (GUI) a další informace týkající se vývoje. Kapitola 4 se zabývá vývojem *Pomůcky velitele jednotky požární ochrany*, dozvíme se, jak zvládat interakci s uživatelem, práci s globálním polohovacím systémem (GPS) a vzdáleným uložištěm Dropbox. Na závěr v kapitole 5 se podíváme na funkčnost aplikací a porovnáme výsledný produkt napříč platformami.

2 Multiplatformní vývoj mobilních aplikací

Na trhu s mobilními zařízeními se vyskytuje celá řada operačních systémů. Mezi nejpoužívanější patří Android, iPhone Operating System (iOS), Windows Phone (WP) a dále pak BlackBerry OS [2]. Pro každý z těchto operačních systémů se aplikace většinou vyvíjejí s pomocí odlišných technologií. Zařízení, na kterých mobilní operační systémy běží je celá řada a často se velice liší svým hardwarovým vybavením. To spolu s mnoha dalšími faktory značně komplikuje práci vývojářům, kteří chtějí svoji aplikaci zpřístupnit co nejvyššímu počtu uživatelů, nehledě na čas strávený vývojem aplikací pro jednotlivé platformy a učením se, jak vyvíjet pro danou platformu.

S problémem multiplatformního vývoje se můžeme vypořádat pomocí přístupu, jenž umožňuje „napsat jeden kód spustitelný na všech zařízeních“. Ovšem, najít společné, univerzální řešení není jednoduché. Přináší to s sebou řadu úskalí, ústupků a omezení. Výsledná aplikace může být nesrovnatelně pomalejší než tatáž aplikace napsaná nativně. Proto je třeba zvolit kvalitní framework pro vývoj multiplatformních aplikací, čímž můžeme vývoj výrazně urychlit a zjednodušit.

2.1 Multiplatformní frameworky pro mobilní zařízení

Je těžké držet krok s neustálými změnami a novinkami v operačních systémech, přesto o to desítky multiplatformních frameworků usilují. Některé nástroje mají možnosti značně omezené, ale jiné, mnohdy placené, naopak umožňují napsat kvalitní aplikace výkonem a vzhledem takřka k nerozeznání od nativně vyvíjených. Mezi výborná a frekventovaně aktualizovaná multiplatformní řešení patří např. Xamarin, Marmelade, Adobe AIR nebo Corona SDK. Bohužel vývojáři musí za licenci zaplatit stovky, někdy až tisíce dolarů. Zaměřili jsme se proto na hledání kvalitních bezplatně poskytovaných nástrojů, jež splňují požadavky kladené na vyvíjené aplikace (viz kapitola 3 a kapitola 4). Byly vybrány celkem čtyři neplacené frameworky. Vývoj v nich je založen z velké míry na webových technologiích.

V následující části je popsáno, co frameworky umí, jaké mají výhody či nevýhody a na základě čeho byl zvolen framework použitý při vývoji výsledných aplikací pro Českou asociaci hasičských důstojníků (ČAHD).

2.1.1 PhoneGap

PhoneGap [18] je jeden z nejznámějších multiplatformních nástrojů. Podporuje operační systémy Android, iOS, Windows Phone a BlackBerry a také některé dnes méně rozšířené systémy jako Bada, Symbian, Tizen, Ubuntu Touch nebo Firefox OS.

Aplikace psané ve PhoneGapu jsou hybridní (částečně webové a částečně nativní). Aplikace běží zabalené v nativním webovém kontejneru (UIWebView), díky čemuž mají oproti klasickým webovým aplikacím přístup k nativním funkcím zařízení. K vývoji aplikací ve PhoneGapu se používají hojně rozšířené webové technologie JavaScript, HTML5 a CSS3. GUI aplikace nevyužívá nativních prvků platformy, ale je stylováno pomocí kaskádových stylů (CSS) a zobrazováno v rámci webového pohledu. Aplikace díky tomu

vypadá na všech zařízeních stejně. Tento přístup ovšem může přinést také komplikace, jelikož výsledná aplikace nemusí být certifikována z důvodu nesplnění vzhledových požadavků a standardů specifikovaných pro danou platformu.

PhoneGap nemá vlastní integrované vývojové prostředí (IDE), ale dá se snadno integrovat pomocí knihoven do běžně používaných vývojových prostředí – Visual Studio, Eclipse a XCode. Výsledná aplikace se musí zkompilovat ve vývojovém prostředí, které odpovídá dané platformě. Aplikace neběží nativně, ale pouze přes prohlížeč, proto může být pomalejší a musí se tedy dbát na efektivně napsaný kód, optimalizaci a profilování.

2.1.2 Appcelerator Titanium

Titanium [24] je dalším často používaným frameworkem. Podporuje méně platforem než předcházející PhoneGap, a sice iOS, Android, BlackBerry a Tizen. Kromě mobilních aplikací umí vygenerovat i webové stránky optimalizované pro mobilní telefony, tzv. mobilní web. Do budoucna je naplánováno přidání podpory vývoje aplikací na Windows 8 a Windows Phone 8. Podle posledních informací by se tak mělo stát zhruba někdy ve čtvrtém čtvrtletí roku 2014 [28].

Vývoj je založen plně na JavaScriptu. Komunitu má Titanium opravdu velmi rozsáhlou, v roce 2013 se jednalo o 500 000 registrovaných uživatelů. Titanium má vlastní IDE (Titanium Studio vycházející z Eclipse), dále umožňuje testování i debugování a pro spouštění programů používá nativní emulátory daných platforem. Výhodou Titania je, že využívá nativních funkcí platforem (geolokace, databáze, práce se sítí, práce se souborovým systémem, . . .) přímo – nikoliv přes prostředníka jako v případě PhoneGapu. JavaScriptový kód je překládán téměř 1:1 do nativního, díky čemuž jsou aplikace rychlejší než u hybridních řešení.

2.1.3 MoSync

MoSync [14] je dalším neplaceným, volně dostupným frameworkem. Kromě tří v dnešní době nejpoužívanějších platforem (Android, iOS, Windows Phone 7.8) podporuje i celou řadu již zastaralých platforem jako např. Windows Mobile. MoSync zatím nepodporuje přímo Windows Phone 8, ale WP 8 je zpětně kompatibilní s předchozími verzemi OS, tzn. umožňuje spouštět i aplikace napsané pro WP 7.8.

Vývoj probíhá v HTML5, JavaScriptu nebo v C/C++, a to podle toho, jaký druh aplikace zvolíme. Na výběr máme ze tří možností:

- vývoj v HTML5 a JavaScriptu – podobně jako u PhoneGapu aplikace fungují na principu webového pohledu,
- hybridní aplikace psané v HTML5 a JavaScriptu, které využívají nativních funkcí,
- plně nativní aplikace psané v C++, HTML5 a JavaScriptu.

Vývojové prostředí MoSync IDE je integrováno do Eclipse IDE. MoSync IDE má vlastní emulátor (pro Android a iOS), ale používá i emulátory nativní. MoSync neumí přímo sám vytvořit výslednou aplikaci na iOS, ale dokáže vytvořit hotový předpřipravený XCode

projekt, který stačí zkopírovat na Mac a spustit v XCode. K nevýhodám MoSyncu patří, že nemá moc rozsáhlou komunitu a v některých oblastech oproti ostatním frameworkům zaostává, např. co se týče přizpůsobování se novinkám a novým verzím OS.

2.1.4 Rhodes

Rhodes [21] je open-source framework od firmy Motorola Solutions, který podporuje vývoj pro iOS, Android a Windows Phone 8. Rhodes využívá architektury Model-view-controller (MVC). Model a Controller jsou psány pomocí Ruby a Views jsou tvořeny v CSS, HTML a JavaScriptu. Dále v sobě Rhodes má zabudovanou podporu ORM (objektově relačního mapování), čímž zjednodušuje práci s databázemi a daty. Ke Rhodesu je také k dispozici kvalitní dokumentace.

Motorola nabízí sadu vývojových nástrojů v balíku RhoMobile Suite, jenž se skládá z částí RhoElements, Rhodes, RhoConnect, RhoHub a RhoGallery. Jeho součástí je i RhoStudio – vývojové prostředí založené na Eclipse, které disponuje RhoSimulatorem umožňujícím spouštět aplikace. Lze použít i běžné emulátory či debugovat přímo na zařízení. Zdrojový kód je kompilován do Ruby byte kódu a posléze interpretován RubyVM interpretem. Tento způsob není zcela nativní, ale nativnímu se přibližuje a je výkonnější než použití WebView. Rhodes je zaměřen hlavně na tvorbu komplexnějších, podnikových aplikací.

2.2 Appcelerator Titanium

Ze čtyř výše uvedených frameworků byl pro vývoj vybrán Appcelerator Titanium. Klíčovým faktorem při rozhodování bylo, že využívá nativních systémových ovládacích prvků, efektivně překládá zdrojový kód do nativního a podporuje všechny požadované funkce (GPS, komunikace po síti, práce s rozšiřitelným značkovacím jazykem (XML), ...). Užitečné je, že má Titanium vlastní IDE (což usnadňuje instalaci a údržbu), navíc má rozsáhlou komunitu, dobrou dokumentaci, spoustu výukových tutoriálů a knih a časté aktualizace. Nevýhodou Titania je chybějící podpora platformy Windows Phone, ale vzhledem k tomu, že požadavkem bylo udělat aplikaci především na Android a iOS, splňuje tento framework vše potřebné.

Titanium nepatří k frameworkům, které umožňují naprosto stejný kód spustit všude. Spíše by se dalo říct, že kód se musí přizpůsobit všude. Business logika, práce se sítí, databází a událostmi je veskrze všude kompatibilní, ale především prvky uživatelského rozhraní se musí kvůli zásadním odlišnostem upravit podle platformy. Appcelerator uvádí, že přenositelnost kódu uživatelského rozhraní (UI) je okolo 80-100 % [23].

Primárním doporučením pro architektonický návrh aplikací v Titaniu je psát modulární aplikace s využitím CommonJS modulů. Titanium na svých stránkách uvádí řadu osvědčených praktik, jak s těmito moduly správně zacházet [5]. CommonJS jsou oddělené nezávislé bloky, díky kterým odpadnou starosti s globálními proměnnými a jmennými konflikty. JavaScript má totiž omezený rozsah platnosti proměnných, jež jsou buď globální, anebo lokální (uvozené klíčovým slovem „var“) v rámci funkce. Existují i jiné

způsoby vývoje v Titaniu, ty jsou ale mnohem méně efektivní a méně výkonné než moduly CommonJS, kterých se používá i v jiných prostředích založených na JavaScriptu, jako je např. Node.js.

Titanium má ve svém jádře zabudovanou podporu JSONu, čehož se velmi často využívá pro práci s daty, parametry funkcí a proměnnými. Ukázka vytvoření okna a předání parametru funkci pomocí JSONu viz výpis 1.

```
var window = Ti.UI.createWindow({
      title  : "Vase cesty k bezpeci",
        layout : 'vertical',
        height : '100%',
        width : '100%',
        backgroundColor : 'white',
      exitOnClose : true
    });
```

Výpis 1: Vytvoření okna a předání parametru funkci pomocí JSONu

Adresářová struktura projektu v Titaniu se skládá ze dvou hlavních složek – *i18n* a *Resources*. Složka *i18n* může obsahovat lokalizační soubory s řetězci pro snadnou podporu více jazyků. Složka *Resources* obsahuje všechny ostatní soubory se zdrojovým kódem, obrázky, atd. Ve složce *Resources* mohou být podsložky jako *android*, *iphone* nebo *mobileweb*, do kterých se umisťují soubory specifické pro danou platformu, např. obrázek načítací obrazovky v rozlišení, jež odpovídá platformě. Součástí každého projektu jsou dva velmi důležité soubory, bez kterých se žádná aplikace neobejde:

- tiapp.xml – má na starosti konfiguraci aplikace (jméno, verze, ikona, minimální verze software development kit (SDK), . . .).
- app.js – slouží jako spouštěcí soubor celé aplikace.

Funkcionalitu Titania lze dále rozšiřovat pomocí tzv. „Titanium+Plus Modulů“. Moduly jsou distribuovány především pomocí Appcelerator Marketplace. Mnohé jsou k dispozici zdarma, ale za některé pokročilejší moduly se musí platit, např. nejdražší z nich *Beacons Module* stojí v současnosti 500$ [4]. Existuje rovněž možnost napsat si moduly vlastní. Návod, jak postupovat při vytváření vlastních modulů, lze najít v dokumentaci Titania [9].

3 Vývoj aplikace Vaše cesty k bezpečí

Inspirací pro aplikaci *Vaše cesty k bezpečí* se stal informačně vzdělávací projekt *Vaše cesty k bezpečí aneb chytré blondýnky radí* [26] Hasičského záchranného sboru Jihomoravského kraje, krajského ředitelství policie Jihomoravského kraje a Diecézní charity Brno. Tento projekt přináší tipy „chytrých blondýnek", které radí, jak se chovat při hrozícím nebezpečí, mimořádných událostech nebo ohrožení Vaší bezpečnosti. V rámci projektu vznikla informační brožura, jež je k dispozici na stránkách Hasičské záchranné stanice Brno.

Na VŠB byl v rámci projektu *Nové technologie ochrany životního prostředí před negativními následky pohybujících se přírodních hmot* [16] vytvořen software *PortaDOCs - Portable Documents* [20]. Jedná se o souhrn nástrojů a prostředků pro tvorbu elektronických dokumentů a jejich zobrazení, a to nejen na mobilních zařízeních. S pomocí této technologie byly digitalizovány dokumenty výše zmíněného projektu *Vaše cesty k bezpečí* a také vznikl mobilní web [27] uzpůsobený pro následnou vizualizaci těchto digitalizovaných dokumentů. Pro většinu moderních mobilních zařízení je mobilní web dostupný přes webový prohlížeč. Webová aplikace je však závislá na internetu a může tedy běžet pomaleji než aplikace mobilní – záleží na rychlosti internetového připojení a odezvě webového serveru.

Úkolem vyvíjené aplikace je v zadaném formátu a stylu a zároveň přehledným způsobem prezentovat třicet tři témat (digitalizovaných dokumentů) z oblasti ochrany obyvatelstva, požární prevence a bezpečnosti občanů. Použijeme-li mobilního zařízení (bez nutnosti připojení k internetu), měla by aplikace umožnit snadný přístup k informacím o tom, jak správně postupovat v případě běžného ohrožení, jakým je např. ohrožení bezpečnosti či majetku v důsledku kriminality.

3.1 Zdroj dat

Aplikace čerpá informace ze zdrojových dat primárně psaných a sbíraných pro výše zmíněnou webovou aplikaci. Data jsou uložena v XML [10] verze 1.0 s kódováním UTF-8. XML dokumenty se skládají z elementů a atributů dohromady tvořících stromovou strukturu, jejíž návrh je znázorněn na obrázku 1 [20]. Ve zdrojových dokumentech se ovšem nevyskytují všechny elementy a některé v návrhu zamýšlené atributy nemají využití. Pro tuto aplikaci byly proto implementovány pouze ty elementy a atributy, které jsou v dokumentech používány. V návrhu na obrázku 1 chybí elementy pro zvýrazňování textu (tučné písmo, hypertextový odkaz, kurzíva), ale v dokumentech se přesto hojně vyskytují. Seznam všech používaných elementů je i s jejich popisem v tabulce 1 [20].

Kromě souborů s dokumenty patří ke zdrojům ještě soubor *catalugue.xml*, který slouží výhradně jako seznam všech dokumentů – jejich názvů a cest k místu uložení. Vychází se z něj při tvorbě GUI – načítají se z něj data pro hlavní obrazovku celé aplikace. XML elementy vyskytující se v souboru *catalugue.xml* jsou sepsány v tabulce 2.

Element	Popis elementu
List	Seznam
ListItem	Položka seznamu
Text	Text
Strong	Tučný text
Link	Hypertextový odkaz
Description	Textový popis psaný kurzívou
Image	Obrázek
Table	Tabulka
TableData	Data, kterými bude plněna tabulka *Table*
DataRow	Datový řádek v tabulce *Table*
DataItem	Datová buňka tabulky v *DataRow*
Section	Logické ohraničení části dokumentu

Tabulka 1: Seznam použitých elementů v dokumentu [20]

Element	Popis elementu
Books	Seznam knih
Book	Seznam všech dokumentů
Document	Digitalizovaný dokument

Tabulka 2: Seznam použitých elementů v katalogu

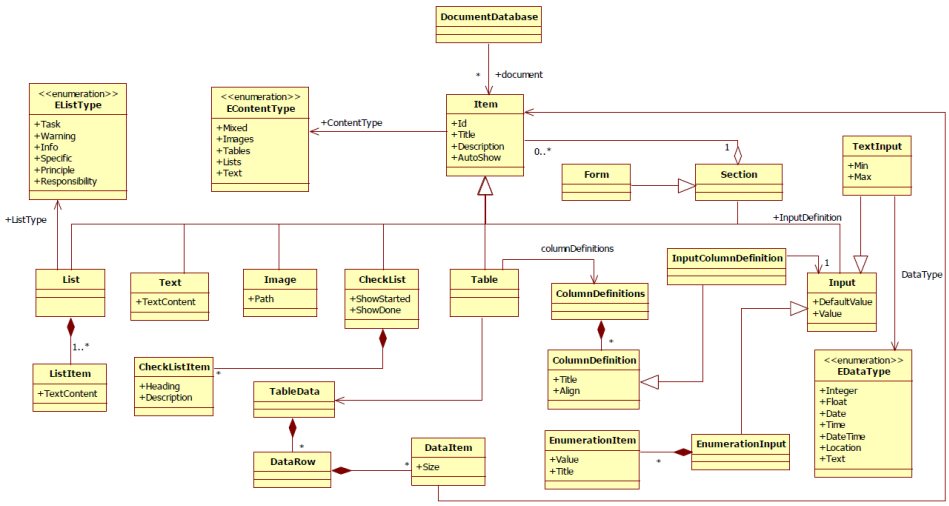
3.2 Dynamické generování prvků GUI

Princip dynamického generování GUI spočívá v tom, že GUI není pevně dáno při psaní aplikace, ale je vytvářeno za chodu. Potřebné informace jsou načteny z přiložených souborů do paměti, na jejich základě jsou vytvořeny komponenty GUI a ty jsou nakonec zobrazeny uživateli. Celý proces je poněkud zdlouhavější a náročnější na výpočetní výkon oproti situaci, kdy jsou všechny potřebné informace známy dopředu a pevně zakotveny v aplikaci. Výhodou je oddělení dat od implementace, čímž je umožněna využitelnost i pro jiné dokumenty v budoucnu.

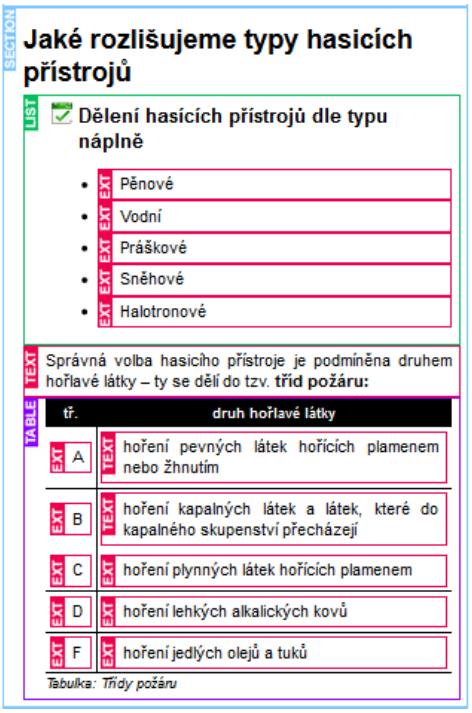
3.2.1 Zpracování XML

Zdrojové XML soubory s daty jsou integrovány do mobilní aplikace a na jejich základě je vytvářeno GUI. Webová aplikace, pro kterou byly XML soubory napsány, používá pro tvorbu GUI XSD a XSLT transformace. Příklad toho, jak by výsledný vzhled GUI měl zhruba vypadat, je vidět na obrázku 2. Jedná se o vizualizaci zdrojových dat s pomocí vzorové šablony, jež využívá právě XSD a XSLT.

Titanium výše zmíněnými technologiemi pro práci s XML nedisponuje, tudíž je třeba použít jiné alternativy. Titanium má v sobě pro práci s XML zabudovanou implementaci funkcí objektového modelu dokumentu (DOM) [7], který odpovídá specifikaci mezinárodního webového konsorcia (W3C). Pro mobilní web Titanium implementuje rozhraní



Obrázek 1: Objektový návrh struktury XML souborů [20]



Obrázek 2: Vzorový vzhled podle šablony [20]

pro programování aplikací (API) DOM verzi 3, ale pro mobilní platformy bohužel pouze základní specifikaci DOM verze 2 s některými nestandardními rozšířeními.

Zpracování XML dat v aplikaci probíhá ve stručnosti následovně:

1. načtení XML do paměti,
2. převod do DOM reprezentace,
3. předání metodě z modulu *collectWalk*,
4. průchod elementy a filtrování,
5. zpracování elementů.

XML je nejprve načteno ze souboru, poté je převedeno do objektové reprezentace DOM voláním funkce *Ti.XML.parseString*. Dále je třeba XML postupně projít. API v Titaniu pro DOM2 nemá zabudovaný žádný efektivní způsob procházení XML, na rozdíl od verze DOM3, která implementuje XPath. Pro procházení a zpracování DOM modelu je v aplikaci napsán modul *collectWalk*, jenž zpřístupňuje funkce *catalogueWalk* a *documentWalk*. Tyto funkce přebírají jako vstupní parametr XML uzel a kontejner View, do kterého jsou přidávány prvky GUI (*documentWalk* navíc přebírá ještě parametr *path* obsahující cestu k souboru). Funkce *catalogueWalk* a *documentWalk* fungují obdobně, ale slouží pro jiný druh XML souboru. Všechny uzly jsou nejdříve odfiltrovány funkcí *acceptNode*, aby se eliminovaly prázdné uzly a procházely se pouze uzly typu *ELEMENT_NODE* a *TEXT_NODE*. Ukázka kódu funkce *acceptNode* viz výpis 2. Následně jsou načteny atributy uzlu, vyhodnotí se typ XML elementu a na základě tohoto typu se element předá dál ke zpracování příslušnou metodou. S každým elementem je třeba zacházet odlišně, protože některé elementy potřebují před svým přidáním projít své potomky a jiné elementy se naopak musí zpracovat ještě před svými potomky.

Ve výpisu 3 je vidět ukázka vyhodnocení typu XML Elementu a jeho předání k dalšímu zpracování.

```
var acceptNode = function(/*XmlNode*/node) {
    var TYP = node.getNodeType();
    var FILTER_ACCEPT = 0;
    var FILTER_SKIP = -1;
    if (node == null) {
        return FILTER_SKIP;
    }
    switch (TYP) {
        case 1:  //ELEMENT_NODE
            return FILTER_ACCEPT;
        case 3: //TEXT_NODE
        if (node.nodeName == '#text' && node.textContent.replace(/\n/g, '').replace(/ /g, ' ') == "")
            {
            return FILTER_SKIP;
        } else{
            return FILTER_ACCEPT;
        }
        default:
```

```
            return FILTER_SKIP;
        }
    };
```

Výpis 2: Filtrování XML uzlů

```
switch(it.tagName) {
  case "heading":
   case "text" :
      ui.text( it , node, view);
      break;
   case "section" :
      ui.section( it , view, source, node, buttonsView, menuData, rootTitle);
      break;
   case "image" :
      var att = getAttributes( it );
      ui.image(it, view, source, node, att);
      break;
   case "listItem" :
      documentWalk(it, view, source);
      break;
   case "list" :
      ui.list (node, it , view, source, menuData, rootTitle, buttonsView, rootTitle);
      break;
```

Výpis 3: Část switche vyhodnocujícího typ XML elementu

3.2.2 Multiplatformní techniky a doporučení pro práci s GUI

Než přejdeme k detailům implementace GUI, měli bychom si stručně shrnout možnosti Titania. Zaměříme se zejména na tvorbu vzhledu aplikací, které podporují co největší množství platforem a zařízení. Pro implementaci GUI Titanium nabízí tyto možnosti [23]:

- větvení podle identifikátoru platformy,
- JavaScript Style Sheets (JSS) soubory,
- JavaScript (JS) soubory specifické pro platformu,
- globální styl,
- Alloy MVC Framework.

Větvení na základě identifikátoru platformy lze použít nejen pro oddělení implementace GUI, ale obecně i pro oddělení jakéhokoliv kódu specifického pro určitou platformu. Existuje několik metod, jak identifikovat platformu, nejčastěji se používá metoda *Ti.Platform.osname* vracející zkrácený identifikátor platformy, např. *ipad* nebo *iphone*, *android*, atd. Větvení se doporučuje použít, pokud je kód rozsáhlý. Další možností, jak oddělit vzhled komponent od kódu, je pomocí tzv. JSS souborů. V JSS souborech se odkazuje na komponenty podobně jako v CSS souborech při psaní webových stránek. Styl

pro jeden JS soubor se píše do jednoho JSS souboru. Např. pro soubor *label.js* by se jeho styl napsal do souboru *label.jss*, a pokud bychom chtěli styl komponenty odlišný podle platformy, stačí jej rozdělit do souborů *label.android.jss* a *label.iphone.jss*. JSS se doporučuje používat pro jednodušší aplikace a pro aplikace nevyužívající CommonJS (implementace JSS nefunguje dobře s moduly CommonJS). Pro složitější aplikace se doporučuje použití tzv. „JS souborů specifických pro platformu“, kdy se styl specifický pro platformu napíše přímo do definice komponenty v JS souboru. Jinou alternativou pro složité aplikace je tzv. „Tweetanium technika“ – napsat objekt, do kterého se uloží globální styl a podle platformy se přiřadí komponentám GUI. Za zmínku stojí také Alloy [1], což je relativně nový MVC framework pro Titanium SDK, jenž umožňuje rozdělení kódu do vrstev. Zatím není příliš rozšířen a jeho dokumentace není tak podrobná jako u Titania.

Vzhledem k tomu, že je při implementaci používáno CommonJS modulů, kód typický pro platformu je oddělen pomocí větvení a souborů specifických pro platformu.

Dobrým předpokladem pro dosažení stejného vzhledu na mnoha platformách a zařízeních je správné pozicování a rozložení komponent GUI. V Titaniu se pro jednotlivé obrazovky využívá komponenty okna *Ti.UI.Window* (dále jen *Window*). Komponenty se přidávají buď přímo do okna, nebo do kontejneru *Ti.UI.View* (dále jen *View*) a ten se následně přidá do *Window*. Rozložení komponent v rámci *View* i *Window* se stanovuje pomocí atributu *layout*. Možnosti rozložení jsou následující:

- Absolutní – umístění komponenty je specifikováno pomocí nastavení atributů *top*, *bottom*, *left*, *right* a *center*.
- Vertikální – komponenty jsou rozloženy odshora dolů.
- Horizontální – komponenty kontejneru jsou rozloženy zleva doprava.

Kontejner *View* se dá pozicovat absolutně nebo relativně vůči svému rodiči (nadřazenému prvku ve kterém je umístěn) a ostatním okolním komponentám. Relativní pozicování je v případě této aplikace daleko vhodnější. Rozměry *View* (šířku a výšku) je vhodné specifikovat procentuálně vůči dostupnému místu, anebo pomocí konstant:

- Ti.UI.FILL – rozměr *View* je přizpůsoben na maximální velikost rodiče.
- Ti.UI.SIZE – rozměr *View* se přizpůsobí velikosti jeho obsahu.

3.2.3 Implementace prvků GUI

V této části se podíváme na implementaci komponent GUI podle XML elementů ve zdrojových souborech. Nebudeme popisovat detailně všechny použité komponenty, ale pouze ty, jejichž implementace byla náročnější nebo se u ní vyskytly nějaké komplikace.

Komponenty jsou přidávány do kontejneru View dynamicky během procházení XML reprezentace modulem *collectWalk*. Přesněji funkce z modulu *collectWalk* mají na starosti procházení a funkce z modulu *ui* vytváření jednotlivých komponent GUI. Jelikož dopředu není známo, kolik místa budou komponenty zabírat, je použito jako hlavního kontejneru na komponenty *ScrollView*. Všechny komponenty jsou postupně přidávány do *ScrollView*,

které je po svém naplnění přidáno do okna (*Window*). *ScrollView* se dá vertikálně scrollovat (posouvat), čímž je zajištěno, že ať je obsah dokumentu jakkoliv dlouhý, vždy bude zobrazeno vše.

3.2.3.1 Element Book Element Book obsahuje elementy Document, jejichž seznam je zobrazen na hlavní obrazovce aplikace – zdrojem dat je soubor *catalogue.xml*. Ukázka hlavní obrazovky je na obrázku 3. Seznam je zobrazen pomocí komponenty *TableView*, jednotlivé řádky jsou vytvářeny pomocí vlastního rozložení, jež je definováno v metodě *tableAdapter* z modulu *ui*. Do komponenty *TableViewRow*, reprezentující jeden řádek, je kromě titulku dokumentu ukládána i cesta k němu. Celý řádek má z estetických důvodů na pozadí nastavený barevný gradient. Velikost řádků se přizpůsobuje jejich obsahu. *TableView* využívá prvku vyhledávání a umožňuje filtrovat řádky podle titulku. Při kliknutí na řádek se vytvoří, naplní a otevře nové okno s dokumentem, který přísluší danému řádku.

Starší verze Androidu měly problém s tím, že se výška *ScrollView* na hlavní obrazovce přizpůsobovala nesprávně a tedy ne tak, jak by podle dokumentace měla. Implementace se proto musela upravit. Výška *ScrollView* se na hlavní obrazovce u starších verzí Androidu musí přibližně dopočítávat. U novějších verzí Androidu už vše funguje tak, jak je uvedeno v dokumentaci.

3.2.3.2 Element Text Text je zobrazován pomocí komponenty *Label*. Aby aplikace vypadala stejně na všech zařízeních, musí být všude i stejná velikost písma, čehož jsme schopni dosáhnout použitím univerzálních, na platformě nezávislých jednotek – density independent pixel (DIP). Jednotka se musí psát pokaždé znovu, jinak se použije defaultní nastavení. Je proto lepší pevně nastavit v konfiguračním souboru projektu *tiapp.xml* atribut *ti.ui.defaultunit* na hodnotu *dip*. Potom už v celé aplikaci nemusíme nikde zadávat jednotku, stačí napsat číslo a vždy se použije jednotka DIP. Pro font bylo zvoleno snadno čitelné bezpatkové písmo *Helvetica Neue*.

V rámci elementu Text se mohou vyskytovat elementy *Strong*, *Description* a *Link*. Jejich účelem je zobrazit zformátovaný text – tučným písmem, kurzívou nebo jako odkaz. S těmito elementy byl zásadní problém, jelikož Titanium umožňuje nastavit pro jeden label pouze jeden formát, tzn. jednotlivé části textu nelze naformátovat zvlášť. Nelze tedy mít jeden *Label* obsahující běžné písmo a např. tučné písmo zároveň. Element *Link* navíc musí fungovat i jako odkaz na webovou stránku.

Jednou možností je rozdělit text na části a přidat každou část textu jako samostatný *Label*. *Labely* jsou pozicovány podle rozložení *View*, ve kterém jsou umístěny, což vede k nehezkému zobrazení – může dojít k tomu, že z ničeho nic je věta ukončena uprostřed řádku a pokračuje až na dalším. Text působí nesouvisle a je špatně čitelný. Kromě *Labelu* je možné text umístit do komponenty *TextArea*, ta má ale stejný problém – umožňuje nastavit pouze jeden formát textu a navíc slouží spíše k zadávání textu než k jeho zobrazování.

Druhou možností je použití komponenty *WebView*. Text se naformátuje pomocí HTML tagů a zobrazí se jako HTML stránka v rámci *WebView*. To s sebou ovšem přináší několik problémů. Tento postup není zcela vhodný vzhledem k velkému objemu textu ve zdrojo-

vých souborech. Často se může vyskytnout několik *WebView* na jeden dokument, což není žádoucí, protože používání mnoha komponent *WebView* v rámci jednoho okna (*Window*) způsobuje delší načítání a obecně se nedoporučuje. U *WebView* se navíc musí zakázat zoom a scrollování a špatně se přizpůsobují jeho rozměry.

Nejlepší možností, jak problém vyřešit, je použití externího open-source modulu *StyledLabel* [22], který je dostupný z Appcelerator MarketPlace a podporuje Android i iOS. Modul umožňuje přidání *Labelu* zformátovaného pomocí HTML tagů obdobně jako formátovaný text při psaní webové stránky. Toto řešení bohužel také způsobuje jisté komplikace, neboť modul pro jednotku písma nepřebírá defaultní nastavení aplikace – DIP. Velikost písma se proto musí přepočítat, aby korespondovala se zbytkem textu.

3.2.3.3 Element Image

Tento element je implementován pomocí komponenty *ImageView*, které se v atributu *image* předává obrázek načtený z disku. Při implementaci se vyskytly dva menší problémy.

Prvním z nich je, že ve zdrojových souborech je cesta k obrázku uváděna relativně. Jelikož se zdrojové soubory nacházejí odděleně v jiných složkách než JS soubory, ve kterých se data zpracovávají, nelze z této relativní cesty poznat, kde se obrázek nachází. Řešením je předat do funkce *documentWalk* i cestu ke zdrojovému souboru, ze kterého jsou data načteny.

Druhý problém souvisí s obrázky, které jsou větší, než je velikost displeje. Takové obrázky se musí zmenšit, aby byly vidět celé. *ImageView* z nějakého důvodu není schopno samo obrázek přizpůsobit tak, aby se vešel na obrazovku při zachování poměru stran. Řešením je zmenšit obrázek pomocí funkce *imageAsResized*. Funkce přebírá dva parametry – šířku a výšku. Nové rozměry se vypočítají ze současných rozměrů a ze šířky displeje zařízení, která je zjištěna při startu aplikace a uložena v aplikační proměnné *Ti.App.SCREEN_WIDTH*. Rozměry obrázku nelze před otevřením okna a provedením finálního rozložení komponent běžným způsobem získat – atributy *width* a *height* vracejí prázdné hodnoty. Obrázek se proto musí převést na tzv. binary large object (BLOB) a z něj už se šířka a výška zjistit dají. Kód provádějící zpracování obrázku je vidět na ukázce 4.

```
if (path != null) {
    var file = Ti.Filesystem.getFile(Ti.Filesystem.resourcesDirectory, path);
    var imageTemp = Ti.UI.createImageView({
        image : file.read(),
    });
    var imageBlob = imageTemp.toBlob();
    var imageWidth = imageBlob.width;
    var imageHeight = imageBlob.height;
    if (imageWidth > Ti.App.SCREEN_WIDTH) {
        var INDENTATION = 10;
        var newWidth = Ti.App.SCREEN_WIDTH - Ti.App.SCREEN_WIDTH / INDENTATION;
        var newHeight = (imageHeight / imageWidth) * newWidth;
        imageTemp.image = imageTemp.image.imageAsResized(newWidth, newHeight);
    }
    view.add(imageTemp);
}
```

Výpis 4: Zpracování obrázku

3.2.3.4 Elementy List a Section Pokud dokument obsahuje členění na *List* a *Section*, na hlavní obrazovce dokumentu jsou zobrazeny speciální tlačítka. Rozdíl mezi nimi je, že *Section* odděluje větší část dokumentu. *List* většinou obsahuje elementy *ListItem*, které jsou zobrazeny pomocí odrážek. Tlačítka *List* a *Section* jsou ve skutečnosti implementovány jako *View*, ale jsou přizpůsobeny, aby fungovaly obdobně jako komponenta *Button* (tlačítko). Do vytvořeného *View*, které má orámovaný okraj, je přidáno *ImageView* s obrázkem a *Label* s titulkem obsaženým v atributu *List* nebo *Section*. *View* reaguje na událost kliknutí, při kterém otevře nové okno s informacemi obsaženými v daném elementu. Po rozkliknutí na další obrazovce už elementy nejsou zobrazovány jako tlačítka, ale je zobrazen pouze titulek dané *Section* nebo *Listu* jako tučně zvýrazněný *Label*.

3.2.3.5 Element Table V Titaniu neexistuje komponenta umožňující zobrazit tabulku o více sloupcích. Návrh zdrojových XML souborů však toto umožňuje a tabulky o více sloupcích jsou v souborech používány. Vzhledem k neexistenci komponenty byla implementace poměrně náročná, jelikož se musela vytvořit obdoba Tabulky. Jednou možností je použít komponentu *TableView* a udělat si vlastní rozložení řádků tak, aby obsahovaly několik *View* (buněk). Tento způsob se nakonec ukázal jako zbytečně komplikovaný. Větší volnost a snazší implementaci přináší vytvoření tabulky pouze s pomocí kontejnerů *View*.

Podle počtu potomků elementu *columnDefinitions* se stanoví, kolik bude sloupců, a vytvoří se hlavička tabulky, která se vzhledem liší od ostatních buňek. Potom se procházejí další elementy a do výsledné tabulky se přidávají obsahy sloupců řádek po řádku.

Jelikož návrh XML souborů je postaven tak, že není stanovena šířka sloupce a chybí informace o tom, který sloupec má větší prioritu, při implementaci mají všechny sloupce prioritu stejnou. Pokud jsou např. zadány čtyři sloupce, každý má šířku 25 %. Speciální výjimkou z tohoto pravidla je případ, kdy jsou sloupce pouze dva –levý sloupec pak bude široký 30 % a pravý 70 % . Je to kvůli zdrojovým souborům, které jsou přiloženy k aplikaci a v kterých se některé tabulky nezobrazovaly hezky.

S tabulkou nastávají problémy při použití více než dvou či tří sloupců, obzvláště pokud používáme mobilní zařízení na výšku. Dochází k zalamování obsahu buňek, a pokud buňky obsahují dlouhé texty nevypadá to dobře. Řešením by mohlo být adekvátně zmenšit velikost písma, aby se text zobrazoval v pořádku bez zalamování, písmo by ale v některých případech bylo pro uživatele příliš drobné a na mobilním zařízení nečitelné. Lepším řešením by tedy mohlo být přizpůsobit šířku sloupce velikosti jeho obsahu. To však v Titaniu není uskutečnitelné, jelikož nelze zjistit, kolik místa bude zabírat komponenta Label na obrazovce před vytvořením všech komponent. Neexistuje zde ani žádná funkce, která by určovala, kolik místa zabere text s daným fontem. Spočítat by se to dalo asi velmi těžce, neboť každý font má jinou šířku písma a navíc se v textu používá různé formátování (viz 3.2.3.2).

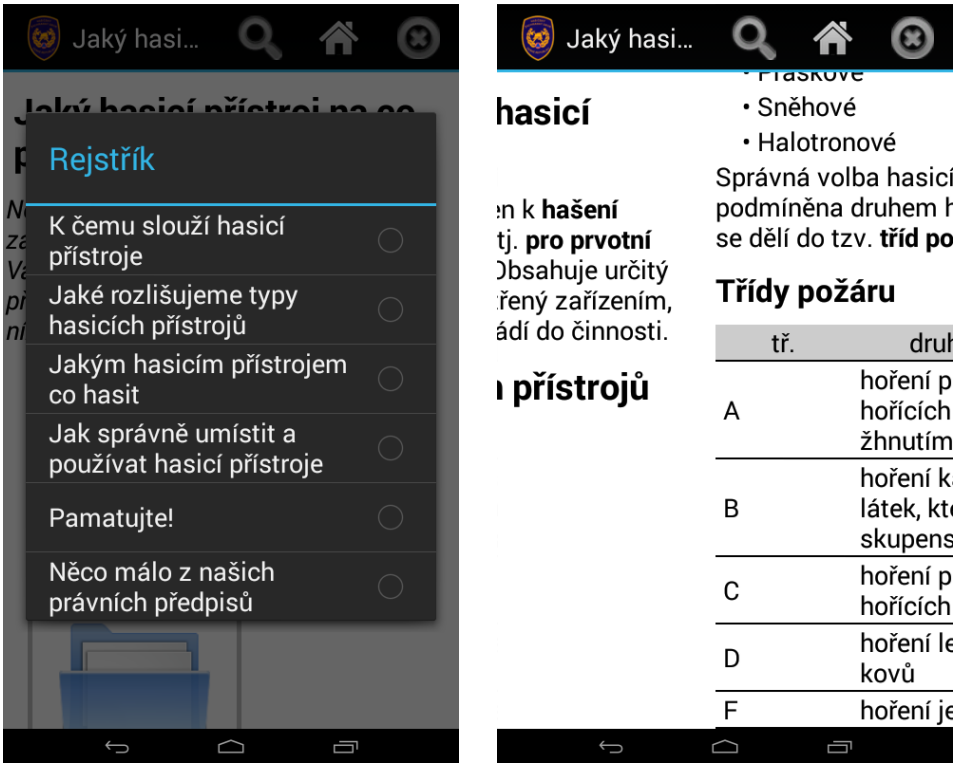
Obrázek 3: Vyhledávání v katalogu na hlavní obrazovce

Při digitalizaci dokumentů by měl být brán zřetel na to, že mnoho sloupců se může na mobilních zařízeních špatně zobrazovat. Místo tabulky by se tudíž měla zvážit jiná komponenta pro zobrazování údajů.

3.3 Ukázka výsledné aplikace

V této části je popsána výsledná aplikace z pohledu jejího použití uživatelem. Na hlavní obrazovce je vidět seznam všech dokumentů. Seznamem můžeme scrollovat nebo jej lze filtrovat s pomocí vyhledávání v horní části obrazovky. Ukázka vyhledávání v seznamu je vidět na obrázku 3. Po kliknutí na položku seznamu se otevře nové okno s vybraným dokumentem.

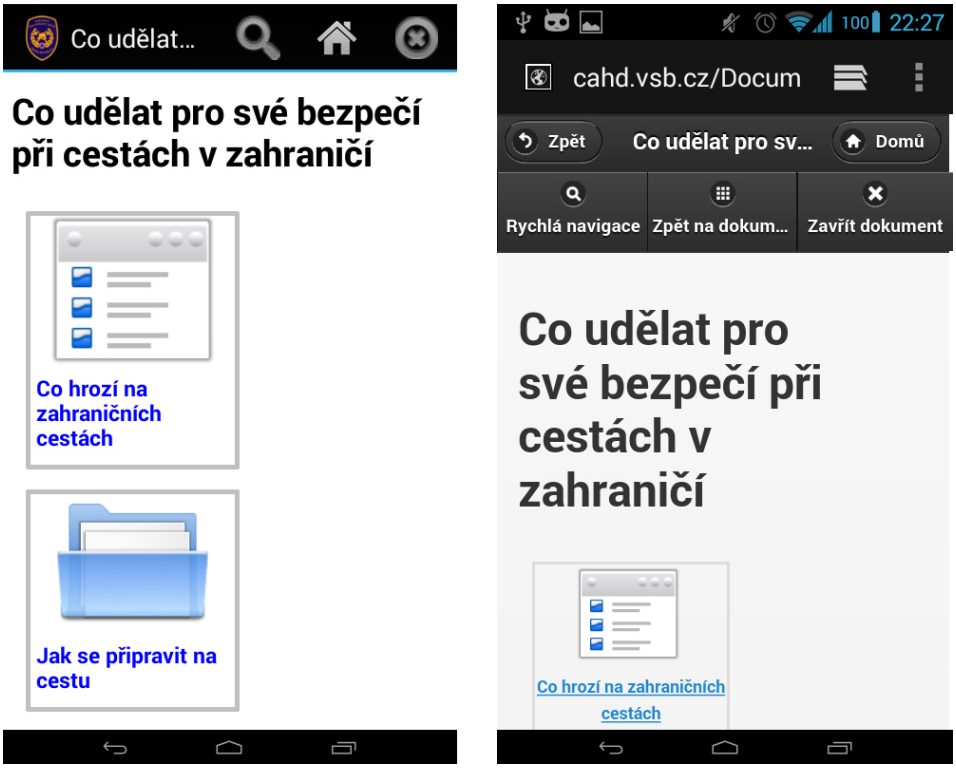
Na obrazovce dokumentu (viz obrázek 5 vlevo) vidíme titulek dokumentu společně s několika tlačítky graficky zobrazenými jako složka a seznam. Po kliknutí na ně se otevře nové okno s informacemi obsaženými v konkrétní sekci dokumentu. V horní části aplikace jsou vidět tlačítka menu. Tlačítko s ikonou „lupy“ zobrazí rychlou navigaci (viz obrázek 4 vlevo), což je další možnost, jak přejít na vybranou sekci dokumentu. Ikona „domu“ vrací na hlavní obrazovku dokumentu. A poslední ikona „křížku“ zavře aktuální dokument.



Obrázek 4: Rychlá navigace (vlevo), horizontální scrollování (vpravo)

Na další obrazovce jsou už vidět konkrétní informace k dané problematice. Obrazovku můžeme vertikálním scrollováním posouvat a pročítat si tak dokument. Pro navigaci napříč obrazovkami a pro návrat na předchozí obrazovku nám poslouží tlačítko zpět, případně můžeme využít tlačítek menu popsaných výše, nebo použít nejjednodušší varianty horizontálního scrollování, které je ukázáno na obrázku 4 vpravo.

Pro srovnání se můžeme podívat, jak vypadá obrazovka stejného dokumentu na výsledné mobilní aplikaci a na webové aplikaci vyvíjené na VŠB v rámci projektu *Nové technologie ochrany životního prostředí před negativními následky pohybujících se přírodních hmot*. Na obrázku 5 vlevo je mobilní aplikace a vpravo webová aplikace. Díky použití nativních komponent zařízení se získalo místo navíc a na obrazovku se vejde více informací.



Obrázek 5: Srovnání mobilní (vlevo) a webové aplikace (vpravo)

4 Vývoj aplikace Pomůcka velitele jednotky PO

Aplikace *Pomůcka velitele jednotky požární ochrany* má na rozdíl od aplikace předcházející sloužit výhradně jako „rychlý“ nástroj a nápověda pro velitele a ostatní příslušníky hasičského sboru na místě zásahu. Použité metodiky [19] *Pomůcky velitele jednotky* vznikly v rámci České asociace hasičských důstojníků za využití grantu Ministerstva vnitra České republiky.

Cílem je zobrazit metodiky a postupy, jak si počínat v krizových situacích, a dále umožnit vyplňování kontrolních listů přímo v centru dění. Vyplněné kontrolní listy se uloží a později (např. po skončení zásahu a návratu na stanici) se odešlou do vzdáleného uložiště.

Pracuje se celkem se třemi dokumenty, z nichž dva obsahují pokyny o tom, jak postupovat při dopravní nehodě a signalizaci. Dostupný je zde i jeden kontrolní list určený pro záznam údajů o detekci nebezpečné látky.

Pomůcka velitele jednotky vychází z podobných dat jako dříve uvedená aplikace *Vaše cesty k bezpečí*. Její data byla rovněž digitalizována s využitím nástroje PortaDOCs v rámci projektu *Nové technologie ochrany životního prostředí před negativními následky pohybujících se přírodních hmot* (viz kapitola 3). Zdrojové XML soubory obou aplikací mají identickou stromovou strukturu (viz obrázek 1), obě aplikace jsou vzhledově totožné, ovládají se stejně a umí stejné věci. Pomůcka velitele jednotky tedy rovněž dokáže zpracovat a zobrazit všechny XML elementy uvedené v tabulce 1. Oproti souborům z aplikace *Vaše cesty k bezpečí* obsahují zdrojové dokumenty navíc XML elementy sepsané v tabulce 3 [20]. Tyto značky se týkají především prvků umožňujících vstup od uživatele.

Z celkového pohledu na funkcionalitu by se dalo říci, že se jedná o aplikaci Vaše cesty k bezpečí rozšířenou o následující sadu funkcí:

1. interaktivní režim,
2. snímání pozice GPS,
3. ukládání do vzdáleného uložiště.

V nadcházejících částech textu jsou tato rozšíření popsána blíže.

4.1 Interaktivní režim

Interaktivní režim se vyskytuje pouze v kontrolních listech a od uživatele je v něm vyžadována určitá interakce – např. vyplnění vstupních polí a uložení. Formát ukládání kontrolních listů je popsán v kapitole 4.3.1. Kontrolní listy nejsou nijak zvláštně označeny, proto je interaktivní režim od běžného dokumentu rozpoznán na základě existence XML elementů z výše zmíněné tabulky 3. K dispozici máme dva druhy interaktivních prvků – vstupní pole a seznamy úkolů. Nyní si je podrobněji popíšeme a podíváme se na řešení jejich implementace.

Element	Popis elementu
InputColumnDefinition	Definice sloupce tabulky, který bude obsahovat vstupní pole
Input	Pole pro vstup uživatele
TextInput	Textové vstupní pole
EnumerationInput	Výčtové vstupní pole
EnumerationItem	Položka výčtu ve vstupním poli *EnumerationInput*
Form	Logické ohraničení formuláře
CheckList	Seznam úkolů
CheckListItem	Položka seznamu úkolů

Tabulka 3: Seznam použitých elementů v aplikaci Pomůcka velitele jednotky PO [20]

4.1.1 Element Input

Nejběžnějším elementem je pole pro vstup. Vyskytuje se především v podobě dvou elementů - *TextInput* a *EnumerationInput*. Textových polí pro vstup je definováno více druhů. Ve zdrojových souborech je zvoleno, jaký druh textu bude v poli zadáván. Celkově se tedy mohou vyskytnout tyto možnosti zadávaných typů:

- celá čísla,
- desetinná čísla,
- prostý text,
- datum,
- čas,
- datum a čas,
- GPS souřadnice,
- výčet možností.

Podle toho, jaký druh textu se zadává, je adekvátně nastaven typ klávesnice, která se uživateli zobrazí. Tak například při zadávání čísel je nabídnuta pouze klávesnice numerická. Jelikož na zadávání data a času existuje komponenta *Titanium.UI.Picker*, není třeba používat klávesnici. Tato komponenta se zobrazí přes celou obrazovku po kliknutí do textového pole. Pro zadávání data se používá typ *Pickeru Titanium.UI.PICKER_TYPE_DATE* a pro zadávání času *Titanium.UI.PICKER_TYPE_TIME*. Tato možnost je ovšem dostupná pouze na Androidu. Systém iOS umí *Picker* zobrazit pouze v rámci *View*, ale vzhledem k nedostatečnému prostoru a horizontálnímu rozložení u tabulky tato možnost zobrazení nepřipadá v úvahu a datum se tedy musí zadat ručně na klávesnici. Naopak výhodou iOS je, že umí zobrazit komponentu pro zadání času i data zároveň – *PICKER_TYPE_DATE_AND_TIME*. Tato komponenta není na Androidu podporována. Datum se zadává ve formátu *DD.MM.RRRR* a čas ve 24-hodinovém kontinentálním formátu. Žádná

speciální komponenta na GPS souřadnice zde není, proto je na zadávání koordinátů použit obyčejný *Titanium.UI.TextField* s numerickou klávesnicí. Poslední situací, která může nastat, je výčet možností. V takovém případě je použita taktéž komponenta *Picker*, ale oproti výše zmíněným ve své standardní, tj. neupravené podobě.

Aby uživatel věděl, co a v jakém formátu do pole zadat, je mu poskytnuta nápověda ve formě zašedlého textu nastaveného v atributu *hintText*. Nápověda je dostupná pouze do doby, než uživatel do daného pole začne psát – toto chování je zakotveno v implementaci komponenty. Pokud měl element výchozí hodnotu nastavenou v atributu *defaultValue*, místo přednastavené nápovědy je nápověda nahrazena touto výchozí hodnotou.

Aby se předešlo výskytu chyb a neplatných údajů, musí být vstupy uživatele ošetřeny proti zadávání neplatných hodnot. Je zde naimplementována řada kontrol. Jedná se o kontroly na nulový vstup, neplatné znaky a neplatné hodnoty přesahující stanovené limity v atributech *min* a *max*. Správnost zadaných GPS souřadnic a časových údajů je kontrolována pomocí regulárních výrazů. Uživatel je na chybu vždy upozorněn patřičnou chybovou hláškou a je mu umožněno neplatný vstup opravit.

Při zobrazení na výšku se tabulkové rozložení pro vstupní pole příliš nehodí, jelikož buňky tabulky se při větším počtu sloupců nevejdou všechny na řádek a musí být hodně zúžené. Dochází také k zalamování textu hlaviček sloupců, jak je popsáno v kapitole 3.2.3.5. Pro mobilní zařízení by bylo lepší zobrazit vstupní pole pod sebou, tedy jedno na řádek, než všechny vedle sebe na stejném řádku.

4.1.2 Element CheckList

Jedná se o speciální seznam, ve kterém si můžeme odškrtnout zahájené a dokončené úkoly. V případě kontrolního listu detekce nebezpečné látky si například můžeme poznačit, zda bylo zabráněno iniciaci. Máme k dispozici dva sloupce, jež ukazují zahájené (*showStarted*), respektive dokončené úkoly (*showDone*). Seznam je implementován pomocí několika komponent *Titanium.UI.Switch*. Klasický zaškrtávací *CheckBox* je dostupný pouze na Androidu a navíc je *Switch* díky větším rozměrům pro aplikaci vhodnější. Na Androidu je *Switch* s popisem ANO/NE, zatímco na iOS je zobrazen jako pouhý přepínač bez popisu.

4.2 Geolokace

Geolokace, neboli identifikace geografické polohy objektu, nám umožňuje získat reálnou geografickou polohu mobilního zařízení. Geolokace bývá úzce spojena s pozicovacími systémy a může sloužit jak k získání přesných geografických souřadnic, tak k získání konkrétní smysluplné adresy. Poloha se dá určovat mnoha způsoby. Mezi nejčastější patří zjišťování pomocí satelitů, mobilního operátora a pomocí internetového spojení [25]. S kontrolním listem se má ukládat i přesná poloha zařízení, je proto žádoucí použít k získání souřadnic GPS v zařízení.

API Titania nám umožňuje získávat GPS souřadnice aktuální pozice dvěma způsoby – buď jednorázovým voláním, nebo nepřetržitým sledováním lokace zařízení pomocí odběru události *location*. Abychom mohli funkce GPS využívat, musí je uživatel naší aplikaci nejdříve povolit. Na iOS se pro tento případ musí nastavit vlastnost *Ti.Geolocation.purpose*

– nastaví se textová hodnota s vysvětlením, za jakým účelem hodláme polohu zařízení využít. Kromě toho, že musíme mít povolen přístup, je k využití GPS také třeba mít aktivované lokační služby. S ohledem na přesnost zaměření musíme brát v úvahu, že větší přesnost vyžaduje vyšší výkon a dochází tak ke zvýšené spotřebě baterie. Na Androidu i iOS máme pro nastavení přesnosti GPS tyto dvě možnosti:

- ACCCURACY_HIGH – největší přesnost na několik metrů,
- ACCCURACY_LOW – menší přesnost - zhruba na kilometr, poloha se získává od mobilního operátora.

Tato nastavení jsou určena pro použití v jednoduchém módu (*simple mode*). Existuje ještě manuální mód (*manual mode*), ale ten se hodí spíše pro pokročilejší práci s GPS, např. pro aplikace sloužící k navigaci. Jednoduchý mód je pro potřeby této aplikace dostačující. iOS implementuje ještě několik dalších nastavení přesnosti, jejichž použití na Androidu není doporučováno.

Pro implementaci byla zvolena nejvyšší přesnost *ACCCURACY_HIGH*. Pro práci s GPS byl napsán modul *geolocation.js*. Po načtení kontrolního listu se k oknu dokumentu připojí funkce z modulu pro GPS. Jakmile je okno otevřeno, zahájí modul svoji práci, tedy za předpokladu, že má uživatel aktivovanou a povolenou GPS. Pokud ano, začne posluchač *locationCallback* nepřetržitě odchytávat polohu. Odchytávání začne ihned od okamžiku otevření okna, jelikož může chvíli trvat, než se pozice zaměří. Pokud uživatel GPS používat nemůže (zařízení GPS nevlastní nebo se nachází v místech odstíněných bez signálu), respektive nechce, je po otevření okna s kontrolním listem dotázán, zda opravdu chce pokračovat a kontrolní list vyplnit bez připojení k GPS, v takovém případě je však kontrolní list stejně jako v případě nedostupnosti signálu z GPS uložen bez záznamu o poloze.

4.3 Datové uložiště

Na místě zásahu je nutné vyplnit kontrolní list a zapsat polohu. Všechny tyto informace je pak třeba někam zaznamenat. Přístup k internetu není pro uložení informací nutný, údaje lze z praktických důvodů ukládat do samotného zařízení. Výsledky je později možno odeslat do vzdáleného uložiště, kde jsou přes internet pohodlně dostupné.

4.3.1 Ukládání výsledků

Výše zmíněné interaktivní komponenty z kapitol 4.1.1 a 4.1.2 reagují na globální událost *eSave*. Stejně tak reaguje na událost i posluchač *locationCallback*, který zaznamenává aktualizace GPS souřadnic. Na konec kontrolního listu je vloženo tlačítko *Uložit*, které událost *eSave* spouští. Po kliknutí na tlačítko se nejprve provedou kontroly vstupních hodnot a posléze se „odpálí“ událost *eSave* a každý posluchač přidá své hodnoty do objektu předávaného v argumentu události. Ukázku implementace posluchače události *eSave* můžeme vidět na výpisu 5. Všechny odběratele události si musíme pamatovat, abychom je mohli po zavření okna smazat, jinak by zůstávaly v paměti spolu s objekty, na které si

udržují reference. Nakonec kód vyvolaný stiskem tlačítka *Uložit* nasbíraná data zpracuje a uloží do textového souboru v adresáři *reports*. Aby se předešlo jmenným konfliktům, je do názvu souboru přidán čas v milisekundách. Název souboru vypadá například takto: „report_1391529816806.txt“.

```
var saveListener = function(e) {
    var item = textField.value;
    if (e.data.length > 0) {
        e.data[e.data.length - 1].data.push(item);
    } else {
        var MainItem = {
            name : "",
            column: [],
            data: []
        };
        MainItem.data.push(item);
        e.data.push(MainItem);
    }
};
Ti.App.addEventListener('eSave', saveListener);
params.callbacks.push(saveListener);
```

Výpis 5: Posluchač události *eSave* předávající hodnotu ze vstupního pole

Formát ukládaného souboru (viz výpis 6) má následující strukturu: na prvním řádku je zapsán časový otisk, na druhém pozice GPS. Dále následují řádky se zadanými hodnotami, a sice jeden údaj na řádek (výjimkou je seznam úkolů). Na začátku každé části je uveden její titulek. Pokud údaj není vyplněn, je místo něj uveden text „PRAZDNE“. Strukturu souboru je v budoucnu možné upravit, aby více vyhovovala potřebám hasičů.

```
Sun Apr 4 2014 16:20:35 GMT+0200 (CEST)
17.1022333 51.7922446 presnost 35.452999114990234
Kontrolni list - Doporuceny postup pri detekci nebezpecne latky
ANO;NE;
ANO;ANO;
Identifikacni udaje konkretni detekce nebezpecne latky
4.4.2014
PRAZDNE
Ostrava
Methan
```

Výpis 6: Formát výsledného souboru (zkráceno pro ilustraci)

4.3.2 Vzdálené uložiště

Vzdáleným uložištěm se myslí uložiště, jež je dostupné odkudkoliv z internetu. Pro ukládání dat se zvažovalo více možností – hledalo se jednoduché řešení se snadnou dostupností pro hasiče a bez nutnosti vkládání zbytečných investic. Přemýšlelo se, zda data zasílat do cloudového uložiště nebo prostřednictvím webové služby na vlastní server. Webová služba by mohla být nakonfigurována pro komunikaci pomocí JSONu. Na straně klienta by se data serializovala a odesílala webové službě, která by data převzala a uložila.

Nevýhodu má toto řešení především z hlediska zajišťování a správy. Server by se musel kvalitně implementovat a následně udržovat. Padlo rozhodnutí, že bude lepší využít hotového cloudového řešení třetí strany. Odpadají tak starostí se zajišťováním hostingu serveru a správy webové služby. Existuje řada webových uložišť, která zajišťují námi požadované služby a nemusí se za jejich využití platit. Mezi nejznámější patří Dropbox, OneDrive a GoogleDrive.

4.3.3 Dropbox

Pro vzdálené ukládání dat byl zvolen Dropbox [8] vzhledem k jeho jednoduchosti, rozšířenosti a všeobecné popularitě. Každý uživatel si může vytvořit svůj vlastní účet, za který v základní variantě neplatí. Místo poskytované zdarma je vzhledem k nízkému objemu dat pro tuto aplikaci dostačující. Dropbox podporuje velké množství platforem a vystavuje své API v mnoha programovacích jazycích.

Titanium však neposkytuje žádnou přímou podporu Dropboxu. V obchodě s moduly je sice nabízeno několik neoficiálních modulů, ale bohužel placených. Nezbývá tedy nic jiného než vlastní implementace komunikace s uložištěm. Dropbox nabízí vývojářům své služby v podobě *Drop-ins* a *Dropbox API*. Drop-ins jsou jednoduché komponenty UI zpřístupňující okamžitý přístup k výběru souboru, tzv. „Chooser“, a rychlé uložení na Dropbox, tzv. „Saver“. Dropbox API poskytuje tři oddělená API. Jedná se o:

1. Sync API – ukládání a synchronizace souborů,
2. Datastore API – synchronizace uživatelských dat mezi více zařízeními a platformami,
3. Core API – čtení, zápis a jiné pokročilejší funkce.

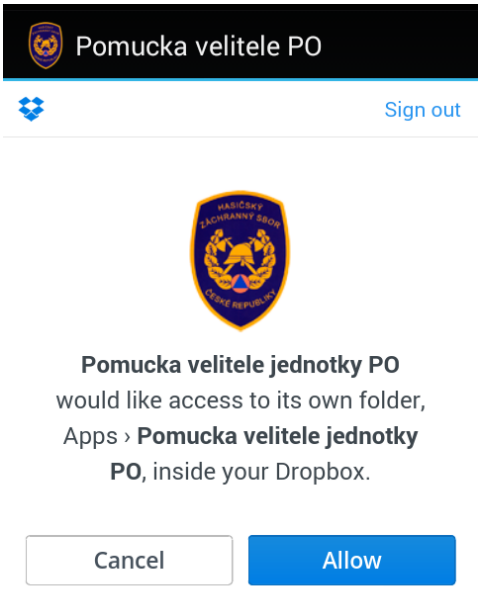
Pro vývoj aplikace v Titaniu se hodí nízkoúrovňová Core API, protože prostřednictvím HTTP protokolu nabízí přímý přístup platformám, které nejsou podporovány oficiálními SDK. K autorizaci uživatele se využívá otevřený standard OAuth 1.0 nebo OAuth 2.0 [17].

Dříve než se může funkcionalita Dropboxu začít využívat, musí se zaregistrovat nová aplikace v panelu App Console [3]. Nejprve se zadá název aplikace, poté popis aplikace a lze nahrát i vlastní ikonu ve dvou velikostech. Registrací aplikace získáme důležité údaje *App key* a *App secret*, které slouží k identifikaci aplikace. Během vývoje může aplikaci testovat až sto uživatelů. Ve chvíli, kdy je aplikace hotová a rozhodneme se ji uvolnit pro veřejnost, musí se odeslat Dropboxu ke schválení.

4.3.4 Implementace komunikace s Dropboxem

Z pohledu uživatele vypadá odesílání uložených kontrolních listů zhruba následovně: na hlavní obrazovce stiskneme tlačítko *Odeslat na server* s obrázkem diskety (na Androidu) nebo tlačítko *Upload* (na iOS). Pokud jsme připojeni k internetu, jsme přesměrováni na přihlašovací obrazovku Dropboxu a po přihlášení jsou soubory automaticky odeslány.

Při implementaci komunikace je použita autorizace OAuth 1.0. Autentifikace je rozdělena do tří kroků [6]. Nejdříve klient odešle POST metodou požadavek */request_token*.



Obrázek 6: Povolení přístupu k Dropbox složce

V odpovědi obdrží *oauth_token_secret* a *oauth_token*. Ve druhém kroku je uživatel přesměrován pomocí GET metody a jednotného identifikátoru zdrojů (URL) */authorize* na webový koncový bod s přihlašovací obrazovkou a oknem pro výběr, zda aplikaci udělit či neudělit přístup (viz obrázek 6). Bez udělení přístupu není možné získat */access_token*, který je obdržen ve třetím kroku voláním metody POST s URL */access_token*. Po úspěšném přihlášení jsou všechny uložené soubory jeden po druhém odesílány metodou */files_put*. Pokud by na Dropboxu už soubor se stejným jménem existoval, bude pro jednoduchost automaticky nahrazen.

Ukázka implementace odesílání souboru viz výpis 7. Implementace byla inspirována volně dostupným kódem ti-dropboxjs [12] a Node.js wrapperem nad Dropbox API [15]. Nejprve si připravíme parametry pro naši žádost. Parametr *overwrite* určuje, že se mají soubory se stejným názvem přepisovat. URL zpráva by měla být podepsána pomocí OAuth. Voláním funkce *Ti.Network.encodeURIComponent* zakódujeme předpřipravené parametry, jsou také odstraněny neplatné znaky přidáním zpětných lomítek. Dále si nachystáme hlavičku zprávy s tím, že na Androidu musíme nastavit parametr *Content-Type*, který se na iOS nechává prázdný. Nakonec vše vložíme do objektu *args*. Uvedeme typ metody, do URL přidáme cestu a kořen, vůči kterému je cesta specifikována (v našem případě „sandbox“), a přidáme tělo zprávy – data ze souboru. Objekt *args* předáme metodě *request*, která se postará o odeslání. V metodě *request* si pomocí funkce *createHTTPClient* vytvoříme klienta, do jehož konstruktoru se uvedou callbacky odchytávající odpovědi. Zavoláním metody *open* na klienta se otevře defaultně asynchronní požadavek a připraví se spojení. Následně pomocí metody *setRequestHeader* nastavíme předpřipravenou hlavičku a zavoláním *send* odešleme tělo zprávy.

```
options = {
    overwrite : true,
    oauth_token : this.accessToken,
    oauth_token_secret : this.accessTokenSecret
};
var oauth1 = require('/lib/oauth');
var params = oauth1(config.app_key, config.app_secret, options);
var urlX = "";
for (var a in params) {
    if (a) {
        urlX += Titanium.Network.encodeURIComponent(a) + '='+Titanium.Network.
            encodeURIComponent(params[a]) + '&';
    }
}
var headers = {
    "content-length" : body.length
};
if (Titanium.Platform.name == 'android') {
    headers = {
        "content-length" : body.length,
        "Content-Type" : "multipart/form-data"
    };
}
var args = {
    "method" : "PUT",
    "headers" : headers,
    "url" : "https:// api-content.dropbox.com/1/files_put/"+root +"/"+escape(path)+"?" + urlX,
    "body" : body
};

request(args, callback);
```

Výpis 7: Vytvoření HTTP zprávy pro odeslání souboru

Do budoucna by bylo lepší neposílat všechny soubory, ale složku v zařízení synchronizovat se složkou na Dropboxu. Můžeme ověřovat existenci souborů na webu a odesílat jen soubory nové. Další možností je udržovat si číslo revize souboru a nechat tak synchronizaci na samotném Dropboxu, který soubor v uložišti přepíše, jen pokud je novější revize. V neposlední řadě bychom mohli také nechat rozhodnutí na uživateli a nabídnout mu, které soubory odeslat a které nikoliv.

5 Testování aplikací na zařízeních

Vzhledem k tomu, že na PC pod OS Windows lze v Titaniu vyvíjet aplikace pouze pro Android (pro vývoj iOS aplikací je nutné vlastnit Mac), byly obě aplikace vyvíjeny primárně pro Android. Během vývoje se však počítalo s budoucí podporou provozu aplikace na iOS a kód byl psán s ohledem na požadavky druhé platformy. Aplikace jsou připraveny na dopracování a portaci na platformu iOS.

Testování probíhalo průběžně během vývoje, a to na čtyřech zařízeních (na dvou mobilních telefonech a dvou tabletech) uvedených v tabulce 4. Aplikace jsou odladěny a jsou plně funkční pro Android verze 2.3 a 4.3. Vzhledem k omezenému počtu vlastních zařízení se systémem Android nebylo fyzicky možné otestovat aplikace na všech typech zařízení a verzích OS. Na verzích Androidu vyšších než 2.3 by se aplikace měly přizpůsobit a bez problému fungovat. Pro testování na jiných verzí 4.1 a 4.2 bylo použito Android emulátoru, který je velmi pomalý, dlouho se zapíná a aplikace z Titania se do něj instalují někdy i několik minut.

Na závěr proběhly testy i na iOS, ale pouze na iOS simulátoru, jelikož aktuální podmínky Applu neumožňují nahrání aplikace do zařízení bez zakoupené Apple Developer License (ani za účelem testování). Simulátor iOS oproti emulátoru Androidu naběhne téměř okamžitě a aplikace se s jeho pomocí na Macu vyvíjejí daleko rychleji a pohodlněji. Titanium ve verzi pro Mac navíc umožňuje vývoj pro Android i iOS současně, takže vlastnictví Macu je pro multiplatformní vývoj v Titaniu velkou výhodou.

Při testech bylo zjištěno, že nejpomalejší částí aplikace je vždy vygenerování GUI z XML souboru. Na méně výkonném tabletu *Yarvik TAB250* trvá vytvoření nového okna několik vteřin. Poté co je GUI vygenerováno a okno vytvořeno, pracuje už aplikace plynule a pružně reaguje na doteky uživatele i na výkonnostně nejslabším testovacím zařízení.

Na tabletu s větším rozlišením se lépe pracuje s interaktivním režimem, jelikož tabulka se vstupními poli není zalamována.

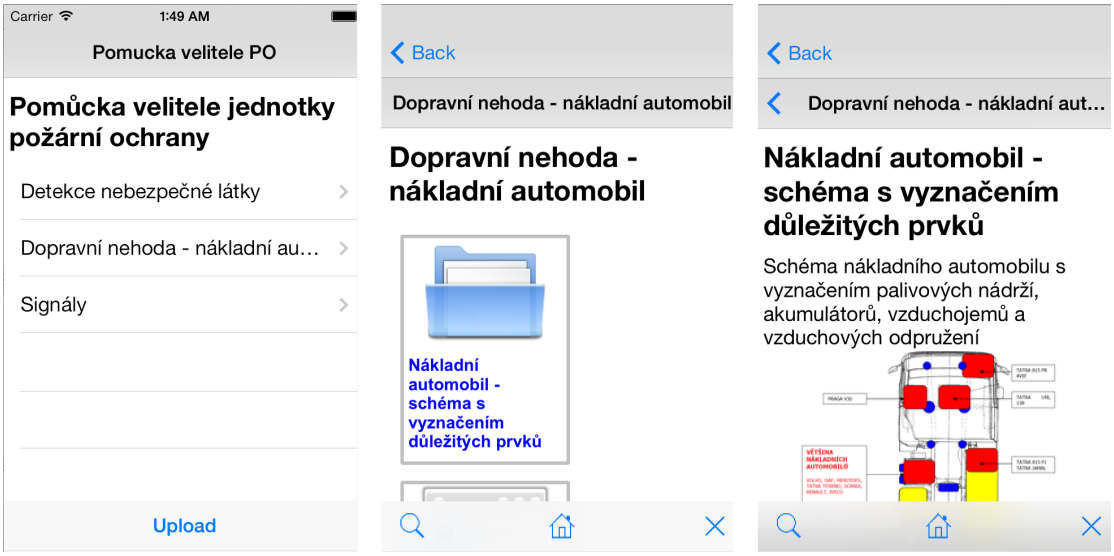
První aplikace Vaše cesty k bezpečí bez větších obtíží funguje i na iOS. Jediným problémem je, že modul StyledLabel použitý na formátování textu neočekávaně odsazuje text a vznikají tak místy delší mezery mezi odstavci v textu. U druhé aplikace Pomůcka velitele PO se na iOS některé komponenty nezobrazují, tak jako v prostředí Androidu. Bude zapotřebí další optimalizace a implementace některých funkcí.

5.1 Srovnání aplikací na různých platformách

Mezi aplikacemi na platformách Android a iOS jsou drobné odlišnosti – pro srovnání se můžeme podívat na obrázky 7 a 8. Za účelem stejné funkcionality se musely naimplementovat odděleně různé komponenty. Pro navigaci mezi okny lze na Androidu používat hardwarového nebo softwarového tlačítka zpět. Na iOS se musí implementovat komponenta *Ti.UI.iOS.NavigationWindow*. Navigačním tlačítkem Back se dostaneme na hlavní obrazovku s katalogem. S pomocí navigace se lze vrátit i o jednu obrazovku zpět, jak je vidět na obrázku 7 vpravo. K zobrazení pomocných tlačítek Rychlá navigace, Zpět na dokument, Zavřít dokument a Upload na server se na Androidu uplatňuje komponenta menu [13] – *Ti.Android.Menu*. Na iOS se na pomocná tlačítka používá odlišná kompo-

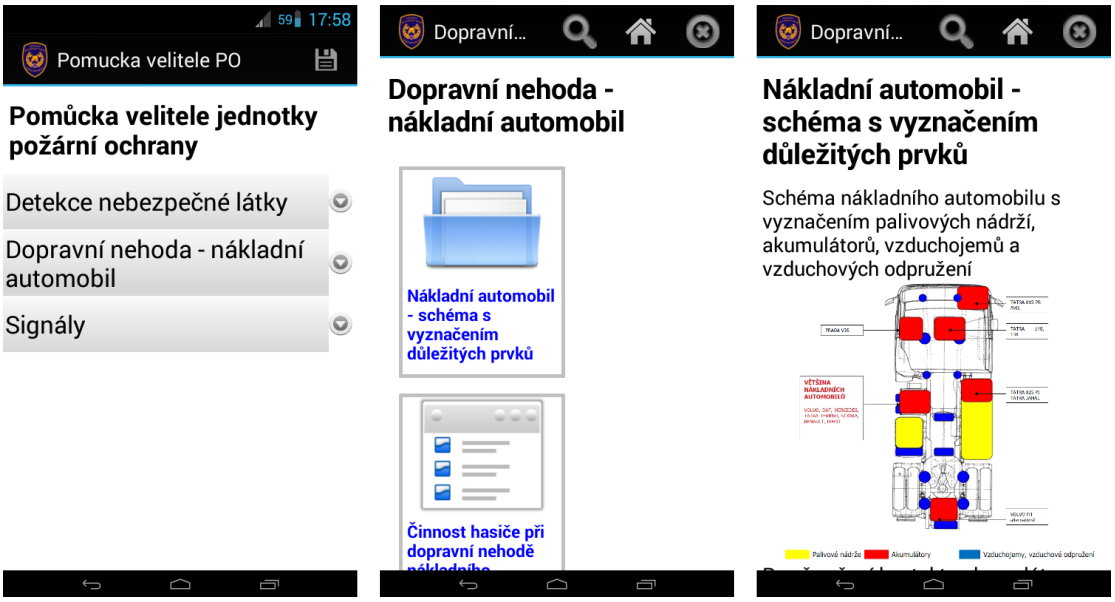
Zařízení	OS	Rozlišení
HTC HD2	Android 4.3	800x480
Sony Ericsson Xperia Ray	Android 2.3.4	840x480
Yarvik TAB250	Android 2.3.3	800x480
Asus Transformer Pad TF701T	Android 4.3	2560x1600

Tabulka 4: Seznam testovacích zařízení



Obrázek 7: Ukázka aplikace na iOS

nenta *Ti.UI.iOS.Toolbar*. Toolbar [11] se na iPhonu vždy zobrazuje v dolní části obrazovky. Jak je vidět na obrázcích, kvůli navigačního panelu a toolbaru se na obrazovce v iOS zobrazí menší množství informací než na Androidu. Mimo tyto odlišnosti se aplikace chovají a ovládají stejně.



Obrázek 8: Ukázka aplikace na Androidu

6 Závěr

Cílem práce bylo nalézt multiplatformní framework pro mobilní aplikace a vytvořit v něm aplikace pro ČAHD. Zvoleným frameworkem byl Appcelerator Titanium a s jeho pomocí vznikly dvě aplikace na Android, které byly napsány tak, aby se v budoucnu snadno daly přizpůsobit i pro provoz na platformě iOS.

Aplikace Vaše cesty k bezpečí se může zpřístupnit pro širokou veřejnost a aplikace Pomůcka velitele jednotky požární ochrany by se v budoucnu ve spolupráci s Fakultou bezpečnostního inženýrství dala rozšířit o další dokumenty nebo kontrolní listy a dále uzpůsobit potřebám České asociace hasičských důstojníků. Mohlo by se vylepšit synchronizování uložených souborů s Dropboxem a zdokonalit rozložení prvků interaktivního režimu. Obě aplikace by se také mohly portovat i na iOS. Výhodou řešení je, že aplikace je možné využít i pro zobrazování jiných dokumentů digitalizovaných v softwaru PortaDOCs.

Přínosem práce pro mě bylo zdokonalení znalosti JavaScriptu, získání zkušeností s multiplatformním vývojem a prohloubení znalostí o operačních systémech iOS a Android.

Jakub Vašica

7 Reference

[1] Alloy Framework. *Appcelerator dokumentace* [online]. [cit. 2014-03-30]. Dostupné z: http://docs.appcelerator.com/titanium/3.0/#!/guide/Alloy_Framework

[2] Android and iOS Continue to Dominate the Worldwide Smartphone Market with Android Shipments Just Shy of 800 Million in 2013, According to IDC. *International Data Corporation (IDC)* [online]. 2014 [cit. 2014-04-25]. Dostupné z: http://www.idc.com/getdoc.jsp?containerId=prUS24676414

[3] App Console. *Dropbox* [online]. [cit. 2014-04-13]. Dostupné z: https://www.dropbox.com/developers/apps

[4] Beacons Module. *Appcelerator Product Market* [online]. [cit. 2014-04-03]. Dostupné z: https://marketplace.appcelerator.com/apps/7619

[5] Coding Best Practices. *Appcelerator dokumentace* [online]. [cit. 2014-03-30]. Dostupné z: http://docs.appcelerator.com/titanium/3.0/#!/guide/Coding_Best_Practices

[6] Core API. *Dropbox* [online]. [cit. 2014-04-13]. Dostupné z: https://www.dropbox.com/developers/core/docs

[7] Document Object Model (DOM). *The World Wide Web Consortium* [online]. [cit. 2014-04-24]. Dostupné z: http://www.w3.org/DOM/

[8] *Dropbox* [online]. [cit. 2014-04-14]. Dostupné z: https://www.dropbox.com/

[9] Extending Titanium Mobile. *Extending Titanium Mobile* [online]. [cit. 2014-04-03]. Dostupné z: http://docs.appcelerator.com/titanium/3.0/#!/guide/Extending_Titanium_Mobile

[10] Extensible Markup Language (XML). *The World Wide Web Consortium* [online]. [cit. 2014-04-24]. Dostupné z: http://www.w3.org/XML/

[11] IOS Human Interface Guideline: Toolbar. *Apple developer* [online]. [cit. 2014-04-23]. Dostupné z: https://developer.apple.com/library/ios/documentation/userexperience/conceptual/mobilehig/Bars.html#//apple_ref/doc/uid/TP40006556-CH12-SW4

[12] SAUNDERS, Aaron K. Javascript dropbox API for Titanium Appcelerator. In: *GitHub* [online]. 2012, 23. 1. 2013 [cit. 2014-04-23]. Dostupné z: https://github.com/aaronksaunders/ti-dropboxjs

[13] Menus. *Android Developer* [online]. [cit. 2014-04-23]. Dostupné z: http://developer.android.com/guide/topics/ui/menus.html

[14] *MoSync* [online]. 2004 [cit. 2014-04-03]. Dostupné z: http://www.mosync.com/

[15] WHITTEN, Brock. NodeJS SDK for Dropbox API. In: *GitHub* [online]. 2012, 25.4.2013 [cit. 2014-04-13]. Dostupné z: https://github.com/sintaxi/node-dbox

[16] Nové technologie ochrany životního prostředí před negativními následky pohybujících se přírodních hmot. *Veřejně přístupná data IS VaVaI* [online]. [cit. 2014-04-27]. Dostupné z: http://www.isvav.cz/h11/projectDetail.do;jsessionid=1D4AC147A1155436DCAB4A47A18513A1?rowId=TA01021374

[17] *OAuth* [online]. [cit. 2014-04-26]. Dostupné z: http://oauth.net/

[18] *PhoneGap* [online]. 2014 [cit. 2014-04-03]. Dostupné z: http://phonegap.com/

[19] POMŮCKA velitele jednotky požární ochrany. *SOUHRN METODICKÝCH PŘEDPISŮ pro činnost jednotek požární ochrany* [online]. Prosinec 2013 [cit. 2014-04-13]. Dostupné z: http://metodika.cahd.cz/#pomucka

[20] KOCYÁN, Tomáš a Jan MARTINOVIČ. *PortaDOCs - Portable Documents*. Dostupné z: http://code.google.com/p/portadocs/

[21] Rhodes. *Motorola Solutions* [online]. [cit. 2014-04-03]. Dostupné z: http://www.motorolasolutions.com/US-EN/RhoMobile+Suite/Rhodes

[22] Styled Label Module. *Appcelerator Product Market* [online]. [cit. 2014-04-08]. Dostupné z: https://marketplace.appcelerator.com/apps/4981

[23] Supporting Multiple Platforms in a Single Codebase. *Appcelerator dokumentace* [online]. [cit. 2014-03-30]. Dostupné z: http://docs.appcelerator.com/titanium/3.0/#!/guide/Supporting_Multiple_Platforms_in_a_Single_Codebase

[24] Titanium Mobile Development Environment. *Appcelerator* [online]. 2008 [cit. 2014-04-03]. Dostupné z: http://www.appcelerator.com/titanium/

[25] Titanium.Geolocation. *Appcelerator dokumentace* [online]. [cit. 2014-04-24]. Dostupné z: http://docs.appcelerator.com/titanium/3.0/#!/api/Titanium.Geolocation

[26] Vaše cesty k bezpečí. *Hasičský záchranný sbor Jihomoravského kraje* [online]. 2010 [cit. 2014-03-30]. Dostupné z: http://www.firebrno.cz/vase-cesty-k-bezpeci

[27] Vaše cesty k bezpečí aneb chytré blondýnky radí. [online]. 2013 [cit. 2014-03-31]. Dostupné z: http://cahd.vsb.cz/

[28] FELONEY, Stephen. Windows 8 Support, What's Going On?. In: *Appcelerator Blog* [online]. 2014 [cit. 2014-03-30]. Dostupné z: http://www.appcelerator.com/blog/2014/01/windows-8-support-whats-going-on/

A Obsah přiloženého CD

Na přiloženém CD se nacházejí tyto složky:

- **Text** - obsahuje text bakalářské práce ve formátu pdf,
- **Aplikace** - instalační soubory apk,
- **Vase_cesty_k_bezpeci** - zdrojové soubory k aplikaci Vaše cesty k bezpečí,
- **Pomucka_velitele_jednotky** - zdrojové soubory k aplikaci Pomůcka velitele jednotky PO.